# Problem A. ASCII Automata Art

| | |
|---|---|
| Time limit: | 3 seconds |
| Memory limit: | 512 megabytes |

This problem statement is quite wordy by itself and does not need a legend. You are given a regular expression and your task is to render its corresponding automaton as an ASCII art text drawing following the specification in the problem statement. Please, see examples.

A regular expression in this problem consists of uppercase letters from `A` to `Z`, special characters `+`, `?`, `*`, and parenthesis that are used for grouping. An input to the problem is given by an ⟨*input*⟩ non-terminal of the following BNF grammar:

⟨*input*⟩ ::= ⟨*expr*⟩

⟨*expr*⟩ ::= ⟨*term*⟩ | ⟨*term*⟩ '`|`' ⟨*expr*⟩

⟨*term*⟩ ::= ⟨*atom*⟩ | ⟨*atom*⟩⟨*term*⟩ | ⟨*term*⟩⟨*atom*⟩

⟨*atom*⟩ ::= ⟨*letter*⟩ | '`(`' ⟨*expr*⟩ '`)`' | ⟨*atom*⟩ '`+`' | ⟨*atom*⟩ '`?`' | ⟨*atom*⟩ '`*`'

⟨*letter*⟩ ::= '`A`' | '`B`' | ... | '`Z`'

A regular expression is rendered as an ASCII art picture using the precise rules that are given below. They recursively define a unique representation for each regular expression as a rectangular *box* of characters with the specified number of rows and columns. Empty characters of the representation, including trailing ones, must be filled with spaces.

A ⟨*term*⟩ that consists of a sequence of $n$ uppercase letters is rendered as a box of 3 rows and $4+n$ columns using `+` and `-` characters to render a border on the first and the last rows and columns as shown in the example. The production rule for the ⟨*term*⟩ non-terminal in the grammar is intentionally ambiguous. The longest possible sequence of adjacent ⟨*letter*⟩ non-terminals in the regular expression must be grouped into a ⟨*term*⟩ and rendered as a single box. For example, a ⟨*term*⟩ of '`NERC`' is rendered as the following $3 \times 8$ box:

```
+------+
+ NERC +
+------+
```

A ⟨*term*⟩ that consists of a sequence of ⟨*atom*⟩ non-terminals and ⟨*term*⟩ non-terminals with letters (as described above) is rendered by laying out the constituent boxes left-to-right, aligned vertically to the top, with 2 columns separating adjacent boxes. The height of the resulting box is equal to the maximum height of the constituent boxes. Each pair of adjacent boxes is joined by rendering `->` characters on the 2nd row in the columns between them. For example, a ⟨*term*⟩ of '`N(E)RC`' (consisting of a sequence: ⟨*atom*⟩ of '`A`', ⟨*atom*⟩ of '`(E)`', and a letters-only ⟨*term*⟩ of '`RC`') is rendered as the following $3 \times 20$ box:

```
+---+  +---+  +----+
+ N +->+ E +->+ RC +
+---+  +---+  +----+
```

An ⟨*expr*⟩ that consists of a single ⟨*term*⟩ is rendered as its ⟨*term*⟩.

An ⟨*expr*⟩ that consists of a '`|`'-separated sequence of ⟨*term*⟩ non-literals is rendered by laying out the corresponding ⟨*term*⟩ boxes top-to-bottom, aligned to the left, with a single row separating adjacent ⟨*term*⟩ boxes. The width of the resulting box is equal to the maximum width of the ⟨*term*⟩ boxes plus 6. There are 3 additional columns on the left, and 3 on the right. The first column and the last column use `+` and `|` characters to join the 2nd rows of all the ⟨*term*⟩ boxes from the top to the bottom one, with `+` placed on the 2nd row of each ⟨*term*⟩ box. The 2nd and the 3rd columns on the left and the 3rd-to-last

and the 2nd-to-last columns on the right have `->` characters on the 2nd rows of the corresponding ⟨*term*⟩ boxes. Additionally, shorter ⟨*term*⟩ boxes are connected on the right with extra `-` characters on their 2nd rows. For example, an ⟨*expr*⟩ of '`C|ON|TEST`' is rendered as the following $11 \times 14$ box:

```
   +---+
+->+ C +---->+
|  +---+     |
|            |
|  +----+    |
+->+ ON +--->+
|  +----+    |
|            |
|  +------+  |
+->+ TEST +->+
   +------+
```

An ⟨*atom*⟩ of '`(`' ⟨*expr*⟩ '`)`' is rendered as its ⟨*expr*⟩.

An ⟨*atom*⟩ of ⟨*atom*⟩ '`+`' is rendered as a box of its source ⟨*atom*⟩ with 2 additional rows at the bottom and 6 additional columns (3 on the left and 3 on the right). The first and the last columns, starting with the 2nd row, and the last row are filled with the connecting characters as shown in the example.

- The 2nd row starts with `+->` and ends with `->+` to connect to the 2nd row of the source ⟨*atom*⟩ box.
- The last row starts with `+<-` to represent a backwards edge in the automaton.

For example, an ⟨*atom*⟩ of '`A+`' is rendered as the following $5 \times 11$ box.

```
   +---+
+->+ A +->+
|  +---+  |
|         |
+<--------+
```

An ⟨*atom*⟩ of ⟨*atom*⟩ '`?`' is rendered as a box of its source ⟨*atom*⟩ with 3 additional rows at the top and 6 additional columns (3 on the left and 3 on the right). The first and the last columns (from the 2nd to the 5th row) and the 2nd row are filled with the connecting characters as shown in the example.

- The first row of ⟨*atom*⟩ '`?`' is always empty (filled with spaces).
- The 2nd row ends with `->+` to represent an epsilon-edge in the corresponding automaton.
- The 5th row starts with `+->` and ends with `->+` to connect to the 2nd row of the source ⟨*atom*⟩ box.

For example, an ⟨*atom*⟩ of '`B?`' is rendered as the following $6 \times 11$ box.

```
+-------->+
|         |
|  +---+  |
+->+ B +->+
   +---+
```

An ⟨*atom*⟩ of ⟨*atom*⟩ '`*`' is rendered as a box of its source ⟨*atom*⟩ with 5 additional rows (3 at the top and 2 at the bottom) and 6 additional columns (3 on the left and 3 on the right). The first and the last columns, with the 2nd and the last row, are filled with the connecting characters as shown in the example.

- The first row of ⟨*atom*⟩ '`*`' is always empty (filled with spaces).
- The 2nd row ends with `->+` to represent an epsilon-edge in the corresponding automaton.
- The 5th row starts with `+->` and ends with `->+` to connect to the 2nd row of the source ⟨*atom*⟩ box.
- The last row starts with `+<-` to represent a backwards edge in the automata.

For example, an ⟨*atom*⟩ of '`C*`' is rendered as the following $8 \times 11$ box.

```
+-------->+
|         |
|  +---+  |
+->+ C +->+
|  +---+  |
|         |
+<--------+
```

An $\langle input \rangle$ is rendered as a box that has 6 more columns than the corresponding box of the $\langle expr \rangle$, with 3 additional columns on the left, and 3 on the right. The 2nd row starts with `S->` to represent the starting state of the automaton and ends with `->F` to represent the final state of the automaton. See the example output.

## Input

The input consists of a single line that corresponds to the $\langle input \rangle$ non-terminal of the grammar given the problem statement and has at most 100 characters in length.

## Output

On the first line of the output, write two integers $h$ and $w$ — the height and the width, correspondingly, of the $h \times w$ box that corresponds to the given $\langle input \rangle$. On each of the next $h$ lines, write $w$ characters of the corresponding ASCII art rendering.

## Example

standard input

```
NE?(ER)C++|(IS)*?|(CHA((LL))ENGING)
```

standard output

```
23 57
      +---+               +----+        +---+
S->+->+ N +->+-------->+->+ ER +->+->+->+ C +->+->+->+->F
   |  +---+  |         |  +----+  |  |  +---+  |  |  |
   |         |  +---+  |          |  |         |  |  |
   |         +->+ E +->+          |  +<--------+  |  |
   |            +---+             |               |  |
   |                              +<--------------+  |
   |                                                 |
   |                                                 |
   +->+--------------->+----------------------------->+
   |  |                |                              |
   |  |                |                              |
   |  +->+--------->+->+                              |
   |     |          |                                 |
   |     |  +----+  |                                 |
   |     +->+ IS +->+                                 |
   |     |  +----+  |                                 |
   |     |          |                                 |
   |     +<---------+                                 |
   |                                                 |
   |  +-----+  +----+  +--------+                     |
   +->+ CHA +->+ LL +->+ ENGING +------------------->+
      +-----+  +----+  +--------+
```

# Problem B. Button Lock

Time limit: 3 seconds
Memory limit: 512 megabytes

You are standing in front of the room with great treasures. The only thing stopping you is the door with a push-button combination lock. This lock has $d$ buttons with digits from 0 to $d-1$. Whenever you press a button, it stays pushed down. You can not pop back up just one button, but there is a "RESET" button — pressing it pops up all other buttons. Initially, no buttons are pushed down.

The door instantly opens when some specific set of digits is pushed down. Sadly, you don't know the password for it. Having read the documentation for this specific lock, you found out that there are $n$ possible passwords for this particular lock.

Find the shortest sequence of button presses, such that all possible passwords appear at least once during its execution. Any shortest correct sequence of button presses will be accepted.

## Input

The first line contains two integers $d$ and $n$ ($1 \le d \le 10$; $1 \le n \le 2^d - 1$). Next $n$ lines describe possible passwords. Each line contains a string $s_i$ of $d$ zeros and ones: for all $j$ from 1 to $d$ the $j$-th character is `1` iff the button with the digit $j-1$ must be pushed down.

All strings $s_i$ are different, and each string contains at least one `1`.

## Output

On the first line, print the number $k$ — the minimum number of button presses. On the second line, print $k$ tokens, describing the sequence. Whenever you press a button with a digit, print that digit. Whenever you press "RESET", print "`R`".

## Examples

| standard input | standard output |
|---|---|
| 2 2<br>10<br>11 | 2<br>0 1 |
| 3 4<br>001<br>111<br>101<br>011 | 6<br>2 0 R 1 2 0 |

## Note

In the second example, the sequence `1 2 R 2 0 1` is also possible.

# Problem C. Cactus Not Enough

Time limit: 3 seconds
Memory limit: 512 megabytes

There was no problem about a cactus at the NERC 2020 online round. That's a bad mistake, so judges decided to fix it. You shall not pass to the World Finals 2021 without solving a problem about a cactus!

A *cactus* is a connected undirected graph in which every edge lies on at most one simple cycle. Intuitively, a cactus is a generalization of a tree where some cycles are allowed. Multiedges (multiple edges between a pair of vertices) and loops (edges that connect a vertex to itself) are not allowed in a cactus.

Cher has got a cactus. She calls cactus *strong* if it is impossible to add an edge to it in such a way that it still remains a cactus. But Cher thinks her cactus is not strong enough. She wants to add the smallest possible number of edges to it to make it strong, i. e. to create a new cactus with the same vertices, so that the original cactus is a subgraph of the new one, and it is impossible to add another edge to it so that the graph remains a cactus. Cher hired you to do this job for her. So... it's on you!

## Input

The input consists of one or more independent test cases.

The first line of each test case contains two integers $n$ and $m$ ($1 \le n \le 10^5$, $0 \le m \le 10^5$), where $n$ is the number of vertices in the graph. Vertices are numbered from 1 to $n$. Edges of the graph are represented by a set of edge-distinct paths, where $m$ is the number of such paths.

Each of the following $m$ lines contains a path in the graph. A path starts with an integer number $s_i$ ($2 \le s_i \le 1000$) followed by $s_i$ integers from 1 to $n$. These $s_i$ integers represent vertices of a path. Adjacent vertices in a path are distinct. The path can go through the same vertex multiple times, but every edge is traversed exactly once in the whole test case. There are no multiedges in the graph (there is at most one edge between any two vertices).

The last line of the input after all test cases always contains two zeros. It does **not** define a test case. It just marks the end of the input and does not require any output.

All graphs in the input are cacti. The total sum of all values of $n$ and the total sum of all values of $m$ throughout the input both do not exceed $10^5$.

## Output

For each test case, first output the line with the minimal possible number of additional edges $A$. Then output $A$ lines, each describing one edge as $u_i$ $v_i$, where $u_i$ and $v_i$ are the numbers of vertices to connect. After adding these edges, the resulting graph must be a strong cactus.

## Example

| standard input | standard output |
|---|---|
| 6 1<br>7 1 2 5 6 2 3 4<br>3 1<br>4 1 2 3 1<br>5 2<br>3 1 3 5<br>3 1 2 4<br>7 2<br>6 1 2 3 4 5 3<br>3 6 5 7<br>0 0 | 1<br>1 4<br>0<br>1<br>5 4<br>2<br>1 3<br>6 7 |

# Problem D. Digits

Time limit: 3 seconds
Memory limit: 512 megabytes

Diana loves playing with numbers. She's got $n$ cards with positive integer numbers $a_i$ written on them. She spends her free time multiplying the numbers on the cards. She picks a non-empty subset of the cards and multiplies all the numbers $a_i$ written on them.

Diana is happy when the product of the numbers ends with her favorite digit $d$. Now she is curious what cards she should pick so that the product of the numbers on them is the largest possible and the last decimal digit of the product is $d$. Please, help her.

## Input

The first line contains the integers $n$ and $d$ ($1 \le n \le 10^5$, $0 \le d \le 9$). The second line contains $n$ integers $a_i$ ($1 \le a_i \le 1000$).

## Output

On the first line, print the number of chosen cards $k$ ($1 \le k \le n$). On the next line, print the numbers written on the chosen cards in any order.

If it is impossible to choose a subset of cards with the product that ends with the digit $d$, print the single line with $-1$.

## Examples

| standard input | standard output |
|---|---|
| 6 4<br>4 11 8 2 1 13 | 5<br>1 2 4 11 13 |
| 3 1<br>2 4 6 | -1 |
| 5 7<br>1 3 1 5 3 | -1 |
| 6 3<br>8 9 4 17 11 5 | 3<br>9 11 17 |
| 5 6<br>2 2 2 2 2 | 4<br>2 2 2 2 |

## Note

In the first example, $1 \times 2 \times 4 \times 11 \times 13 = 1144$, which is the largest product that ends with the digit 4. The same set of cards without the number 1 is also a valid answer, as well as a set of 8, 11, and 13 with or without 1 that also has the product of 1144.

In the second example, all the numbers on the cards are even and their product cannot end with an odd digit 1.

In the third example, the only possible products are 1, 3, 5, 9, 15, and 45, none of which end with the digit 7.

In the fourth example, $9 \times 11 \times 17 = 1683$, which ends with the digit 3.

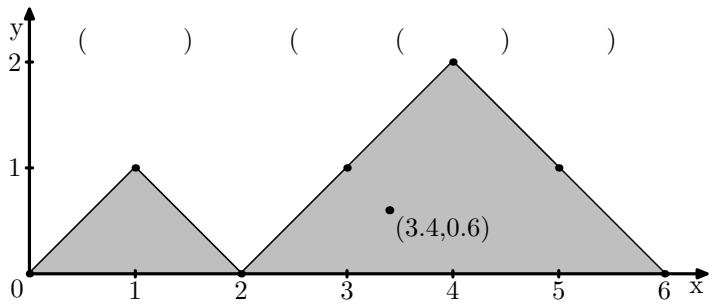
In the fifth example, $2 \times 2 \times 2 \times 2 = 16$, which ends with the digit 6.

# Problem E. Equilibrium Point /\/\

Time limit: 3 seconds
Memory limit: 512 megabytes

Consider a balanced bracket sequence $s$ with one type of brackets: '(' and ')'.

There is a common geometrical representation of such a sequence. Starting at the point $(0, 0)$, you draw a polyline, for each bracket moving along a vector $(1, 1)$ if it is an opening bracket, and along $(1, -1)$ if it is a closing bracket.




Consider an area between this curve and the line $y = 0$. It is a set of polygons. This area has its center of mass at some point $(x, y)$. Note that the center of mass might be outside of the area.

You are to solve the reverse problem. Given the length $n$ and a point $(x, y)$, find any balanced bracket sequence of length $n$ such that the center of mass of its geometrical representation is located at $(x, y)$.

## Input

The first line contains three numbers $n$, $x$, and $y$ ($n$ is an even integer, $2 \le n \le 36$; $0 < x, y < n$) — the length of the desired sequence and the coordinates of the desired center of mass.

It is guaranteed that $(x, y)$ is the center of mass of some balanced bracket sequence of length $n$, with Euclidean-distance error of no more than $10^{-9}$.

## Output

Output a balanced bracket sequence with brackets '(' and ')' of length $n$ such that the center of mass of its geometrical representation is located at the point $(x, y)$, with Euclidean-distance error of no more than $10^{-7}$.

## Example

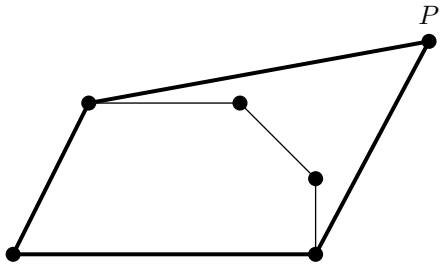
| standard input | standard output |
|---|---|
| 6 3.4 0.6 | ()(()) |

# Problem F. Fiber Shape

Time limit: 3 seconds
Memory limit: 512 megabytes

Imagine a board with $n$ pins put into it, the $i$-th pin is located at $(x_i, y_i)$. For simplicity, we will restrict the problem to the case where the pins are placed in vertices of a convex polygon.

Then, take a non-stretchable string of length $l$, and put it around all the pins. Place a pencil inside the string and draw a curve around the pins, trying to pull the string in every possible direction. The picture below shows an example of a string tied around the pins and pulled by a pencil (a point $P$).




Your task is to find an area inside this curve. Formally, for a given convex polygon $S$ and a length $l$ let's define a *fiber shape* $F(S, l)$ as a set of points $t$ such that the perimeter of the convex hull of $S \cup \{t\}$ does not exceed $l$. Find an area of $F(S, l)$.

## Input

The first line contains two integers $n$ and $l$ ($3 \le n \le 10^4$; $1 \le l \le 8 \cdot 10^5$) — the number of vertices of the polygon $S$ and the length of the string. Next $n$ lines contain integers $x_i$ and $y_i$ ($-10^5 \le x_i, y_i \le 10^5$) — coordinates of polygon's vertices in counterclockwise order. All internal angles of the polygon are strictly less than $\pi$. The length $l$ exceeds the perimeter of the polygon by at least $10^{-3}$.

## Output

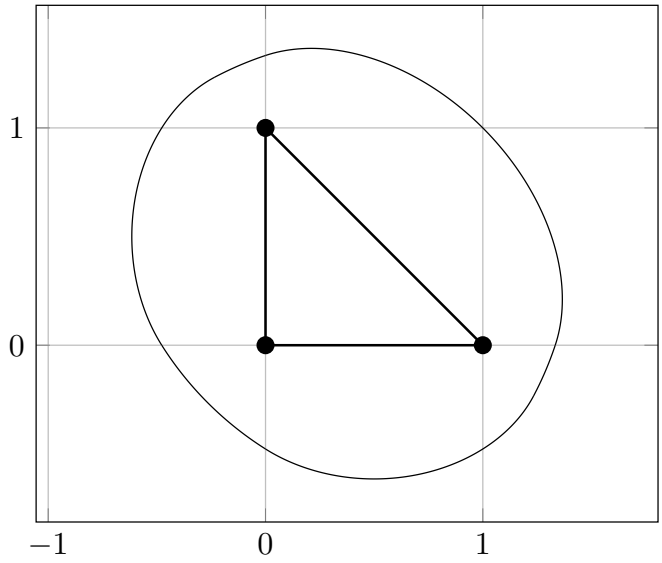
Output a single floating-point number — the area of the fiber shape $F(S, l)$. Your answer will be considered correct if its absolute or relative error doesn't exceed $10^{-6}$.

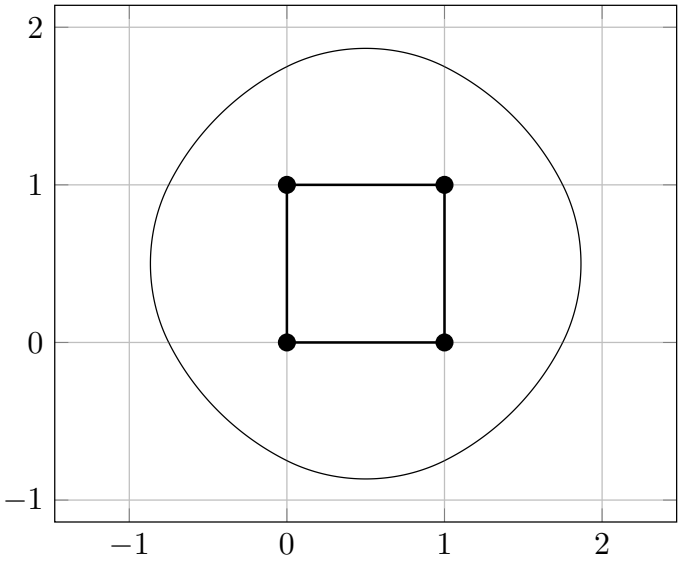
## Examples

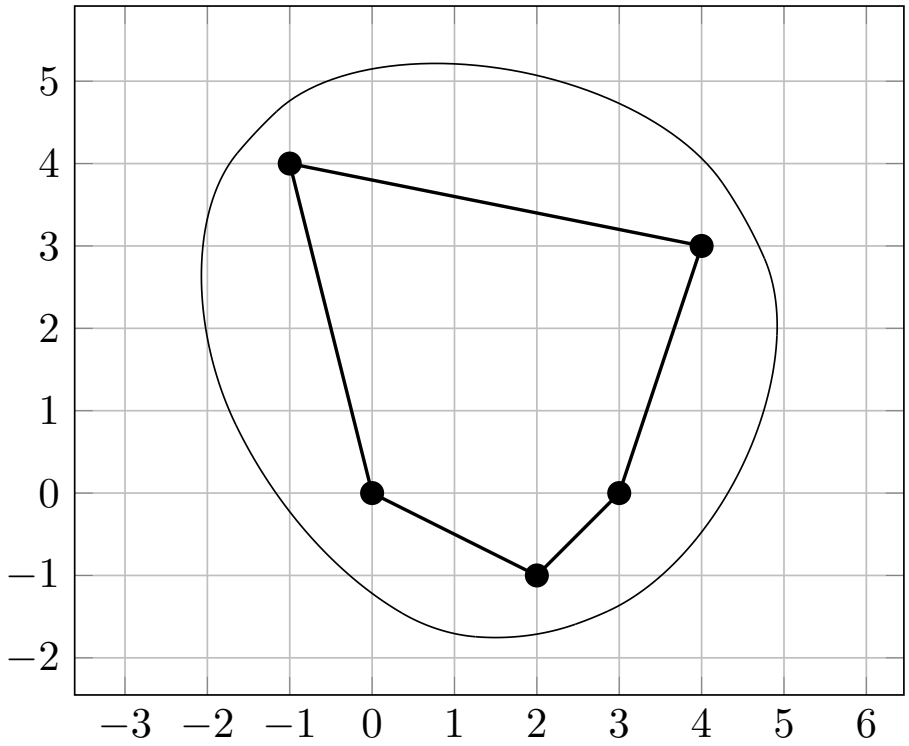
| standard input | standard output |
|---|---|
| 3 4<br>0 0<br>1 0<br>0 1 | 3.012712585980357 |
| 4 5<br>0 0<br>1 0<br>1 1<br>0 1 | 5.682061989789656 |
| 5 17<br>0 0<br>2 -1<br>3 0<br>4 3<br>-1 4 | 37.719371276930820 |

## Note

The following pictures illustrate the example tests.



# Problem G. Guide

Time limit: 3 seconds
Memory limit: 512 megabytes

Mister Gooti is the world-famous guide of The Freezing Isles. The topology of the Isles can be represented as a tree with cities at the vertices and two-way roads between them. Gooti prepares a new sightseeing tour over the Isles. He wants to find the shortest path that starts in the capital and visits $k$ different cities, including the capital. Please, help him.

## Input

The first line of the input contains the number of tests $T$ ($1 \le T \le 100$). Each test consists of two lines. The first line contains the overall number of cities $n$ in the Isles and the requested number of cities $k$ for the tour ($1 \le k \le n \le 100$). The second line contains the description of the tree in a rooted manner: $n-1$ integers where the $i$-th integer, $p_i$, is the parent of the city $i+1$ ($1 \le p_i \le i$). The capital is the city with the number 1 — the root of the tree.

## Output

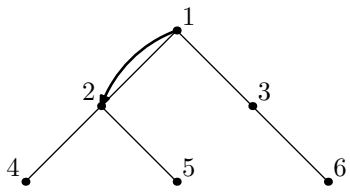
For each test, the first line of the output shall contain the length of the path $l$. The second line shall contain $l+1$ integers — the cities that lie on the path in the order of the traversal.

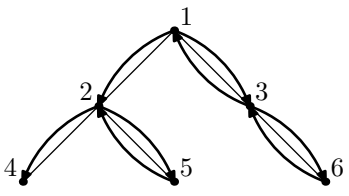
## Example

| standard input | standard output |
|---|---|
| 3<br>6 2<br>1 1 2 2 3<br>6 6<br>1 1 2 2 3<br>6 4<br>1 2 3 4 5 | 1<br>1 2<br>8<br>1 3 6 3 1 2 5 2 4<br>3<br>1 2 3 4 |

## Note

The following pictures illustrate all the three tests from the example.




# Problem H. Hard Optimization

Time limit: 3 seconds
Memory limit: 512 megabytes

You are given a set of $n$ segments on a line $[L_i; R_i]$. All $2n$ segment endpoints are pairwise distinct integers.

The set is *laminar* — any two segments are either disjoint or one of them contains the other.

Choose a non-empty subsegment $[l_i, r_i]$ with integer endpoints in each segment ($L_i \le l_i < r_i \le R_i$) in such a way that no two subsegments intersect (they are allowed to have common endpoints though) and the sum of their lengths ($\sum_{i=1}^{n} r_i - l_i$) is maximized.

## Input

The first line contains a single integer $n$ ($1 \le n \le 2 \cdot 10^3$) — the number of segments.

The $i$-th of the next $n$ lines contains two integers $L_i$ and $R_i$ ($0 \le L_i < R_i \le 10^9$) — the endpoints of the $i$-th segment.

All the given $2n$ segment endpoints are distinct. The set of segments is laminar.

## Output

On the first line, output the maximum possible sum of subsegment lengths.

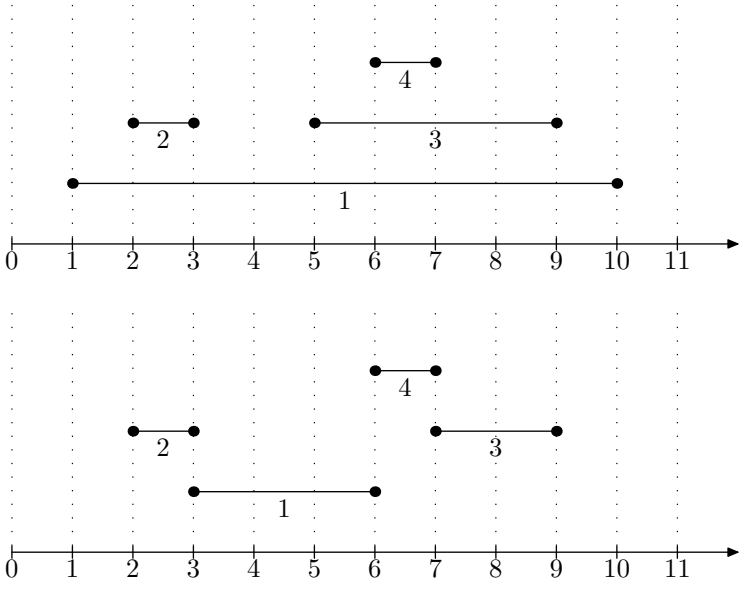
On the $i$-th of the next $n$ lines, output two integers $l_i$ and $r_i$ ($L_i \le l_i < r_i \le R_i$), denoting the chosen subsegment of the $i$-th segment.

## Example

| standard input | standard output |
|---|---|
| 4<br>1 10<br>2 3<br>5 9<br>6 7 | 7<br>3 6<br>2 3<br>7 9<br>6 7 |

## Note

The example input and the example output are illustrated below.



# Problem I. Is It Rated?

Time limit: 3 seconds
Memory limit: 512 megabytes

The popular improv website *Interpretation Impetus* hosts regular improv contests and maintains a rating of the best performers. However, since improv can often go horribly wrong, the website is notorious for declaring improv contests *unrated*. It now holds a wager before each improv contest where the participants try to predict whether it will be rated or unrated, and they are now more popular than the improv itself.

Izzy and $n$ other participants take part in each wager. First, they each make their prediction, expressed as `1` ("rated") or `0` ("unrated"). Izzy always goes last, so she knows the predictions of the other participants when making her own. Then, the actual competition takes place and it is declared either rated or unrated.

You need to write a program that will interactively play as Izzy. There will be $m$ wagers held in 2021, and Izzy's goal is to have at most $1.3 \cdot b + 100$ wrong predictions after all those wagers, where $b$ is the *smallest* number of wrong predictions that any other wager participant will have after all those wagers.

The number $b$ is not known in advance. Izzy also knows nothing about the other participants — they might somehow always guess correctly, or their predictions might be correlated. Izzy's predictions, though, do not affect the predictions of the other participants and the decision on the contest being rated or not — in other words, in each test case, your program always receives the same inputs, no matter what it outputs.

## Interaction Protocol

First, a solution must read two integers $n$ ($1 \le n \le 1000$) and $m$ ($1 \le m \le 10\,000$). Then, the solution must process $m$ wagers. For each of them, the solution must first read a string consisting of $n$ `0`s and `1`s, in which the $i$-th character denotes the guess of the $i$-th participant. Then, the solution must print Izzy's guess as `0` or `1`. Don't forget to flush the output after printing it! Then, the solution must read the actual outcome, also as `0` or `1`, and then proceed to the next wager, if this wasn't the last one.

Your solution will be considered correct if it makes at most $1.3 \cdot b + 100$ mistakes, where $b$ is the smallest number of mistakes made by any other participant. Note that if a solution outputs anything except `0` or `1` for a wager, it will be considered incorrect even if it made no other mistakes.

There are 200 test cases in this problem.

## Example

| standard input | standard output |
|---|---|
| 3 4<br>000<br><br>1<br>100<br><br>1<br>001<br><br>0<br>111<br><br>1 | <br><br>0<br><br>0<br><br>1<br><br>1 |

## Note

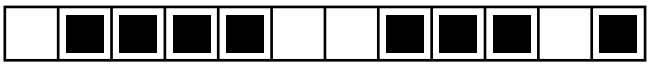
In the example, the participants made 1, 2, and 3 mistakes respectively, therefore $b = 1$ (the smallest of these numbers). Izzy made 3 mistakes, which were not more than $1.3 \cdot b + 100 = 101.3$, so these outputs are good enough to pass this test case (as are any other valid outputs).

# Problem J. Japanese Game

Time limit: 3 seconds
Memory limit: 512 megabytes

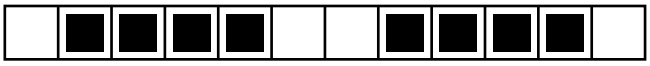
Joseph really likes the culture of Japan. Last year he learned Japanese traditional clothes and visual arts and now he is trying to find out the secret of the Japanese game called Nonogram.

In the one-dimensional version of the game, there is a row of $n$ empty cells, some of which are to be filled with a pen. There is a description of a solution called a *profile* — a sequence of positive integers denoting the lengths of consecutive sets of filled cells. For example, the profile of $[4, 3, 1]$ means that there are sets of four, three, and one filled cell, in that order, with at least one empty cell between successive sets.



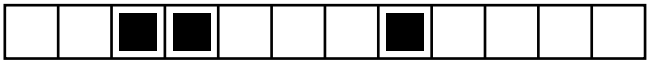
A suitable solution for $n = 12$ and $p = [4, 3, 1]$.

A wrong solution: the first four filled cells should be consecutive.



A wrong solution: there should be at least one empty cell before the last filled cell.

Joseph found out that for some numbers $n$ and profiles $p$ there are lots of ways to fill the cells to satisfy the profile. Now he is in the process of solving a nonogram consisting of $n$ cells and a profile $p$. He has already created a *mask* of $p$ — he has filled all the cells that must be filled in every solution of the nonogram.



The mask for $n = 12$ and $p = [4, 3, 1]$: all the filled cells above are filled in every solution.

After a break, he lost the source profile $p$. He only has $n$ and the mask $m$. Help Joseph find any profile $p'$ with the mask $m$ or say that there is no such profile and Joseph has made a mistake.

## Input

The only line contains a string $m$ — the mask of the source profile $p$. The length of $m$ is $n$ ($1 \le n \le 100\,000$). The string $m$ consists of symbols `#` and `_` — denoting filled and empty cells respectively.

## Output

If there is no profile with the mask $m$, output the number $-1$. Otherwise, on the first line, output an integer $k$ — the number of integers in the profile $p'$. On the second line, output $k$ integers of the profile $p'$.

## Examples

| standard input | standard output |
|---|---|
| __#_____ | 2<br>3 2 |
| _# | -1 |
| ___ | 0 |

# Problem K. King's Task

Time limit: 3 seconds
Memory limit: 512 megabytes

The brave Knight came to the King and asked permission to marry the princess. The King knew that the Knight was brave, but he also wanted to know if he was smart enough. So he asked him to solve the following task.

There is a permutation $p_i$ of numbers from 1 to $2n$. You can make two types of operations.

1. Swap $p_1$ and $p_2$, $p_3$ and $p_4$, ..., $p_{2n-1}$ and $p_{2n}$.
2. Swap $p_1$ and $p_{n+1}$, $p_2$ and $p_{n+2}$, ..., $p_n$ and $p_{2n}$.

The task is to find the minimal number of operations required to sort the given permutation.

The Knight was not that smart actually, but quite charming, so the princess asks you to help him to solve the King's task.

## Input

The first line contains the integer $n$ ($1 \le n \le 1000$). The second line contains $2n$ integers $p_i$ — the permutation of numbers from 1 to $2n$.

## Output

Print one integer — the minimal number of operations required to sort the permutation. If it is impossible to sort the permutation using these operations, print $-1$.

## Examples

| standard input | standard output |
|---|---|
| 3<br>6 3 2 5 4 1 | 3 |
| 2<br>3 4 2 1 | -1 |
| 4<br>1 2 3 4 5 6 7 8 | 0 |

## Note

In the first example, you can sort the permutation in three operations:

1. Make operation 1: $3, 6, 5, 2, 1, 4$.
2. Make operation 2: $2, 1, 4, 3, 6, 5$.
3. Make operation 1: $1, 2, 3, 4, 5, 6$.